K-Automorphism: A General Framework for Privacy Preserving Network Publication *

Lei Zou
Huazhong University of Science and Technology
Wuhan, China
zoulei@mail.hust.edu.cn

Lei Chen
Hong Kong University of Science and Technology
Hong Kong
leichen@cse.ust.hk

M. Tamer Özsu
University of Waterloo
Waterloo, Canada
tozsu@cs.uwaterloo.ca

ABSTRACT

The growing popularity of social networks has generated interesting data management and data mining problems. An important concern in the release of these data for study is their privacy, since social networks usually contain personal information. Simply removing all identifiable personal information (such as names and social security number) before releasing the data is insufficient. It is easy for an attacker to identify the target by performing different structural queries. In this paper we propose k-automorphism to protect against multiple structural attacks and develop an algorithm (called KM) that ensures k-automorphism. We also discuss an extension of KM to handle "dynamic" releases of the data. Extensive experiments show that the algorithm performs well in terms of protection it provides.

1. INTRODUCTION

Social network applications, such as hi5 (hi5.com), facebook (facebook.com), and myspace (myspace.com), have become popular for sharing information. As a consequence, the amount of social network data has grown rapidly, and this offers rich opportunities for data mining and analysis, for example, to find community groups and their evolution [2, 10]. However, social network data usually contain users' private information; it is important to protect these information in any sharing and mining activities. There are well-known examples of unintended release of private information in released (also called published) data, causing organizations to become increasingly conservative in releasing these data sets. Realization of the promise of social networks requires addressing these concerns. Before publishing all these data for analysis, data mining, and other purposes, it is necessary to ensure that the published data will not contain any private information.

In this paper, we focus on the "identity disclosure" problem [13], which is one possible privacy leak concern when a social network is published. Formally, given a published social network G^*, if an adversary can locate the target entity t as a vertex v of G^* with a high probability, we say that the identity of t is disclosed. A naive anonymization method would remove all personal identifiable information before publishing the network, such as names and social security numbers (SSN). However, even when a network is published without any identity information, it is still possible to locate the target with a high probability based on some structural information around the target [7]. In fact, once it can be determined that a published vertex v of G^* corresponds to target t, all sensitive attributes associated with v can be recognized as t's private information. In other words, all the private information about t is released to the adversary, such as the bank account balance in a financial network or infected diseases in a disease spread network. Moreover, based on disclosed identities, some sensitive link information can be easily derived as well. For example, if we can locate two targets t_1 and t_2 in G^*, it is straightforward to figure out whether there is an edge between t_1 and t_2. The relationship between t_1 and t_2 may be sensitive and private. Therefore, "identity disclosure" is a critical factor to consider in publishing privacy-preserved network data.

In order to identify private information in released data, an adversary usually launches different queries using some background knowledge. For example, in tabular data publication, the quasi-identifier attributes (the minimal set of attributes that can be joined with external information to re-identify individual records) can be used as background knowledge. Similarly, any topological structure of the network can be utilized to identify the target in the released network. There can be four types of structural attacks in this environment [7, 23, 13]: degree-attack, sub-graph attack, 1-neighbor-graph attack, and hub-fingerprint-attack. We discuss these in detail in Section 2; the following example demonstrates the problem using one of these types.

Example 1. Given a social network G in Figure 1a, Figure 1b shows the naive anonymized network G' obtained by removing all individuals' names in G. We assume that the target entity is Bob. Note that, the numbers beside vertices in Figure 1 are not vertex labels, but vertex IDs that we introduce to simplify description. From Figure 1b, we can observe that if we know that Bob has four neighbors (i.e. degree attack), we can uniquely identify that Bob is vertex 7 in G'.□

This simple example shows that a naive privacy-preserved published network (one where the identities are removed) is still susceptible to these structural attacks. A number of recent works study privacy-preserving network publication [7, 23, 13]. However, these

*This work was partially done when the first author was visiting University of Waterloo as a visiting scholar. The first author was partially supported by National Natural Science Foundation of China under Grant 70771043. The second author was supported by Hong Kong RGC GRF 611608 and NSFC/RGC Joint Research Scheme N HKUST602 /08. The third author was supported by National Science and Engineering Research Council (NSERC) of Canada.

VLDB '09, August 24-28, 2009, Lyon, France

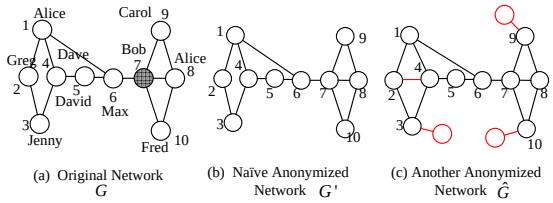


Figure 1: Anonymized Networks

suffer from the following limitations.

1) All but one [7] of the proposals consider only a single type of attack [23, 13]. Furthermore, we do not know of any technique that can guarantee the privacy of a released network under a sub-graph attack (1-neighbor sub-graph attack [23] is a special case of sub-graph attack). In practice, it is not realistic to assume that an attacker would launch only one type of attack (i.e. an adversary has only one type of structural information about the target). One has to assume that there will be simultaneous multiple attacks and the techniques need to be resilient to all of them.

2) The only solution that can handle multiple attacks [7] introduces considerable uncertainty into released networks. Since the released network only contains a summary of structural information about the original network, users have to generate some random sample instances of the released network for further analysis. The samples come from a large number of possible worlds, which introduces much uncertainty, making subsequent analysis difficult.

3) Existing methods do not consider dynamic releases. This is important in evolutionary networks and dynamic social network analysis [2, 10]. For example, given a series of online trading networks, such as eBay (eBay.com), based on community evolution in these networks, we can predict the trend of consumers' purchasing behavior. These applications require republishing data at different times to support dynamic analysis. However, all existing privacy-preserving network publication methods consider only "one-time" release. Even though each released network G_t^* at time T_t can guarantee privacy individually, an adversary can still identify the target with a high probability by collecting the information from multiple releases.

Considering the above limitations, we propose a systematic method for privacy-preserving publishing of social network data. The technique has three advantages.

1) *It can guarantee privacy under any structural attack.* In our method, we do not assume one type of attack. We assume that an adversary can have complete information about the target, such as degree, neighbors, shortest-distances from hubs and so on. Our method can guarantee privacy, even though an adversary can launch multiple and different types of structural attacks.

2) *The released network has no uncertainty.* The released network generated by our method can provide not only a summary of the structural information regarding the whole network, but also structural information about each individual vertex.

3) *It can guarantee privacy under dynamic releases.* Even though an adversary can have historical information about the target, the target cannot be identified in our released networks.

The intuition of our proposed method is the following: Assume that there are k-1 automorphic functions F_a (a=1,...,k-1) in the released network G^*, and for each vertex v, $F_{a_1}(v) \neq F_{a_2}(v)$ ($a_1 \neq a_2$). Thus, for each vertex v in G^*, there are always k-1 other symmetric vertices. This means that there are no structural differences between v and each of its k-1 symmetric vertices. Thus, it is not possible to distinguish v from the other k-1 symmetric vertices using any structural information. Therefore, the target cannot be identified with a probability higher than $\frac{1}{k}$.

Table 1: Meanings of Commonly Used symbols

G	The Original Network
G'	The Naive Anonymized Network
G''	The Anonymized Graph after Partition and Alignment.
G^*	The Anonymized Graph after KM algorithm
G_i^*	The Anonymized Graph after GenID algorithm at time T_t.
U_i	A group of blocks.
P_{ij}	A block in group U_i

However, finding the automorphic functions F_a is a key problem. In this paper, we develop three techniques, namely graph partitioning, block alignment and edge copy.

In order to address the privacy disclosure in dynamic releases, we propose "vertex ID generation" technique. In our method, the intersection of query results in different publications of the same network G always has at least k candidates. Thus the probability of identification is kept at $\frac{1}{k}$.

In summary, we make the following contributions:

1. We propose "k-automorphism" model for privacy preserving network publication, which can guarantee privacy under any structural attack.
2. We propose a systematic method to protect the released social network data from all structural attacks. Specifically, we propose an algorithm to convert original network G into k-automorphic network G^*, which is then released.
3. We consider dynamic releases of networks. In order to avoid privacy disclosure in re-publication of networks, we propose vertex ID generation technique.
4. Extensive experiments and comparisons show the superiority of our methods.

The remainder of the paper is organized as follows. Background knowledge and related work are discussed in Section 2. k-automorphism model is proposed in Section 3. We propose KM algorithm in Section 4. The privacy protection in dynamic releases is discussed in Section 5. We evaluate our method in Section 6. Section 7 concludes the paper.

2. PRELIMINARIES AND RELATED WORK

We first briefly review the terminology that we will use in this paper. Note that, for ease of presentation, we use the terms "graph" and "network" interchangeably, as well as the terms "release" and "publish". Table I lists the symbols used in this paper. For most of the paper, we only consider non-labeled networks. A network $\langle V, E\rangle$ is defined in the usual manner, where V is the set of vertices and E is the set of edges.

DEFINITION 2.1. ***Graph Isomorphism***. *Given two graphs $Q = \langle V_Q, E_Q \rangle$ and $G = \langle V_G, E_G \rangle$, Q is isomorphic to G, if and only if there exists at least one bijective function $f: V_Q \rightarrow V_G$ such that for any edge $(u, v) \in E_Q$, there is an edge $(f(u), f(v)) \in E_G$.*

DEFINITION 2.2. ***Graph Automorphism.*** *An automorphism of a graph $G = \langle V, E\rangle$ is an automorphic function f of the vertex set V, such that for any edge $e = (u, v)$, $f(e) = (f(u), f(v))$ is also an edge in G, i.e., it is a graph automorphism from G to itself under function f. If there exist k automorphisms in G, it means that there exists k-1 different automorphic functions.*

DEFINITION 2.3. ***Sub-Graph Isomorphism***. *Given two graphs Q and G, if there exists at least one sub-graph X in graph G such that Q is isomorphic to X under the bijective function f, graph Q is sub-graph isomorphic to graph G. We call X a* sub-graph match *(match for short) of Q in G. The vertex $f(u)$ in G is called the* match vertex *with regard to vertex u in Q.*

2.1 Structural Attack

DEFINITION 2.4. ***Query***. *Given a social network G, a* query Q *refers to any information that an attacker can use to extract private information from G. The result of Q is a set of vertices $V' \subseteq V$. Each $v_i \in V'$ is called a* match vertex.

DEFINITION 2.5. ***Structural Attack***. *Given a released network G^*, if a query Q over G^* launched by an attacker has a limited number of match vertices in G^*, then target t might be uniquely identified. If Q is based on the structural information about t in G^*, this is called a* structural attack.

The definition of a query Q in this paper is restricted. Obviously, an attacker can also launch a query Q based on non-structural information (such as vertex label) to identify the target. In this paper, we only consider structural attacks. Four types of structural attacks have been identified in network data publication [7, 23, 13]. We illustrate these attacks through Figures 1 and 2.

1) *Degree Attack.* Assume that an adversary knows that Bob has four neighbors. According to the released network in Figure 1b, the adversary can identify Bob uniquely, since only vertex 7's degree is four. Vertex refinement attack [7] is a generalized type of degree attack. It assumes that an adversary can know the number of the target's n-hop neighbors, where $n \geq 1$.

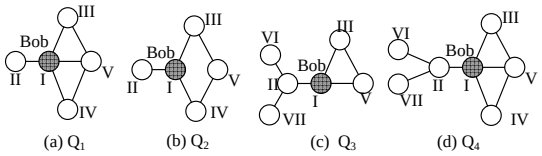


Figure 2: Query Graphs

2) *Sub-graph Attack.* Assume that an adversary knows that query Q_4 (in Figure 2d) exists around Bob, and vertex I in Q_4 corresponds to Bob (as in Figure 1, Roman numerals beside vertices in Figure 2 are vertex IDs, not vertex labels). Since there is only one match of Q_4 in the released network, Bob can be uniquely identified.

3) *1-Neighbor-Graph Attack.* Assume that an adversary knows Bob's neighbors, and the connections among Bob and his neighbors, which is denoted by Q_1 in Figure 2a. There is only one match of Q_1 in the released network. Thus, Bob's privacy will be compromised. As noted earlier, 1-neighbor-graph attack is a special case of sub-graph attack.

4) *Hub Fingerprint Attack.* Assume that some hub vertices have been identified in the released network. If an adversary knows the distances between Bob and these hubs, Bob may also be identified.

2.2 Information Loss and Utility

In privacy preserving data publishing, it is necessary to balance usually conflicting concerns: privacy protection and information loss. Before we discuss our method in detail, we first propose measures of information loss and utility that we employ. In this work, we use the *anonymization cost* to quantify information loss.

DEFINITION 2.6. ***Anonymization Cost.*** *Given an original network G and its anonymized version G^*, the anonymization cost in G^* is defined as*

$Cost(G, G^*) = (E(G) \cup E(G^*)) - (E(G) \cap E(G^*))$

where $E(G)$ is the set of edges in G.

The rationale of using anonymization cost to measure the information loss in G^* is that a lower anonymization cost indicates that fewer changes have been made to the original graph G. Furthermore, some techniques also use statistical network measures to evaluate the utility of released network [23, 13, 7], such as degree difference, average shortest-paths and cluster coefficient. We also use these statistical measures in our experiments.

2.3 Related Work

Privacy-preserving data publication has received increasing interest in database community [7, 13, 15, 16, 18, 19, 23, 9, 14, 6, 4, 3]. Most previous work focus on tabular-data publication. With the increasing popularity of social network applications, analysis of these data has also started to attract attention. How to publish social network data for data mining and analysis without leaking any privacy information is an interesting problem that has been studied [7, 13, 23, 1, 5, 8, 21].

A number of recent works study privacy-preserving network publication [7, 13, 23]. As noted earlier, all of these except one [7] assume that an adversary launches only one type of attack. For example, Zhou and Pei study how to protect released networks from 1-neighbor-graph attack [23], and degree attack is addressed by Liu and Terzi in [13]. However, in practice, an attacker can launch multiple attacks to identify the target and the techniques need to be resilient in the face of multiple attacks.

As an example, given an original network G in Figure 1a, according to existing methods [23, 13], we get an anonymized network $\hat{G}$ in Figure 1c. In $\hat{G}$, there are always at least 2 vertices whose degrees are d, for any d. Furthermore, for any vertex v and its 1-neighbor graph $NG(v)$, there always exists another vertex v', whose 1-neighbor graph $NG(v')$ is isomorphic to $NG(v)$. Consequently, $\hat{G}$ can guarantee privacy under both degree attack and 1-neighbor sub-graph attack. However, if we know that query Q_4 (Figure 2d) exists around target Bob, Bob's identity will still be compromised, since there is only one match of Q_4 in $\hat{G}$.

Hay et al. [7] propose "automorphically equivalent" and discuss multiple attacks. However, the solution introduces considerable uncertainty in released networks. Given the same original network G in Figure 1a, according to their graph generalization method, G is partitioned into two blocks. Each block is condensed into one super node, and the edges between two blocks are condensed into super edges, as shown in Figure 3. The number of edges in the super node X is denoted as X's weight $d(X, X)$, and the number of edges between X and Y is denoted as the super edge (X, Y)'s weight $d(X, Y)$. Although this method can guarantee privacy in the released network, there is much uncertainty in the released network. Since users cannot know the structural information in each supernode, they need to generate samples of the super-graph for further analysis. Authors point out that the number of possible worlds for samples is:

$$|W(G)| = \prod_{X \in V} \binom{\frac{1}{2}|X|(|X|-1)}{d(X,X)} \prod_{X,Y \in V} \binom{|X||Y|}{d(X,Y)}$$

Obviously, the large number of possible worlds ($|W(G)|$) introduces much uncertainty in the released network, which complicates the use of the released data. It also leads to large information loss, such as structural information about each vertex in original networks.

Besides identity disclosure that we focus on, protecting sensitive links among the individuals in the anonymized social network has also received attention. Ying and Wu in [21] study how anonymization algorithms that are based on randomly adding and removing edges change the spectrum of the network. The "link-disclosure" problem they define is orthogonal to "identity disclosure" problem discussed in this paper. Frikken and Golle [5] study how to protect privacy, if a group of authorities can jointly reconstruct a entire

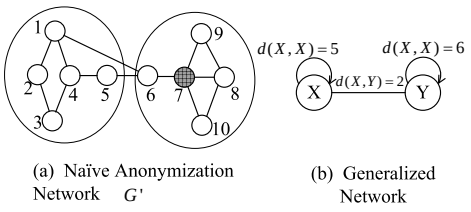


Figure 3: Graph Generation Method

graph G by assembling their own pieces of G. The problem is different from ours, since we only consider centralized data publication.

3. K-AUTOMORPHISM

In order to guarantee privacy from any structural attack, we propose the following concept.

DEFINITION 3.1. ***K-automorphic Network***. *Given a network G, (a) if there exist k-1 automorphic functions F_a (a=1,...,k-1) in G, and (b) for each vertex v in G, $F_{a_1}(v) \neq F_{a_2}(v)$ ($1 \leq a_1 \neq a_2 \leq k-1$), then G is called a* k-automorphic network.

Obviously, if G is a k-automorphic network, for any vertex v in G, we cannot distinguish v from its $k-1$ symmetric vertices based on any structural information. Thus, an adversary cannot identify v from G with a probability higher than $\frac{1}{k}$. Therefore, the problem that we want to solve in this paper is defined as follows:

Given an original network G, find a network G^*, where G is a sub-graph of G^* and G^* is a k-automorphic network. G^* is published as G's anonymized version.

We require that G is a sub-graph of G^*. Thus, all link information in the original network G should be preserved in its anonymized version G^*. Furthermore, in order to minimize anonymization cost (Definition 2.6), and to ensure that the anonymized network is as similar to the original as possible, we require that $|E(G^*) - E(G)|$ is minimized, where $|E(G)|$ is the set of edges in graph G.

In this paper, we propose a systematic method to convert a social network G into a k-automorphic network G^*. Although our method addresses all structural attacks, for clarity, we only use sub-graph attack as an example to show how we protect the privacy in released network data. We prove in Theorems 4.3 and 4.4 that the released network G^* produced by our algorithm is a k-automorphic network, which can resist any structural attack.

Given a sub-graph query Q, if there are k ($k > 1$) matches of Q, no adversary can identify the target uniquely. Recall Example 1, where a naive anonymized network G' was generated by removing all names from the original network G (Figure 1b). Given a sub-graph Q_2 in Figure 2b, there are two matches in G'. Thus, using query Q_2, an attacker cannot identify Bob uniquely. However, k-match is not sufficient to avoid identity disclosure. If these k matches have some shared vertices, the privacy associated with the shared vertices may still be compromised. For example, an adversary may know that query Q_3 in Figure 2c exists around Bob. There are eight matches of Q_3 in the social network G': $(7,6,9,8,1,5)$, $(7,6,9,8,5,1)$, $(7,6,8,9,1,5)$, $(7,6,8,9,5,1)$, $(7,6,10,8,1,5)$, $(7,6,10\ ,8,5,1)$, $(7,6,8,10,1,5)$ and $(7,6,8,10,5,1)$. However, these eight matches have the shared match vertices (i.e. vertices 6,7 in G') with regard to vertex I and II in query Q_3. As shown in Figure 2c, vertex I corresponds to target Bob in query Q_3. In this case, although Q_3 has eight matches, an adversary can still identify Bob in G' uniquely. To overcome the limits of k-match, we define the concept of *different matches* as follows:

DEFINITION 3.2. ***Different Matches.*** *Given a sub-graph query Q and two matches m_1 and m_2 of Q in a social network G', where m_1 and m_2 are isomorphic to Q under functions f_1 and f_2, respectively, if there exists no vertex v (in query Q) whose match vertices in m_1 and m_2 are identical, (i.e. $f_1(v) = f_2(v)$), we say that m_1 and m_2 are* different matches.

Consider two matches $m_1 = (7,6,9,8,1,5)$ and $m_2 = (7,6,9,8,5,1)$ of query Q_3 in Figure 2c. For vertex I in query Q_3, the match vertices of vertex I in m_1 and m_2 are both vertex 7 in G'. Thus, m_1 and m_2 are NOT different. On the other hand, for two matches $m_3 = (7,6,9,10,8)$ and $m_4 = (4,5,1,2,3)$ of query Q_2, there exists no vertex v (in query Q_2) whose match vertices in m_1 and m_2 are identical. Thus, these two matches are different.

Therefore, in order to avoid identifying the target from the released network G^*, we propose *k-different match* principle.

DEFINITION 3.3. ***k-different match principle.*** *Given a released network G^* and any sub-graph query Q, if (a) there exist at least k matches of Q in G^*, and (b) any two of the k matches are different matches according to Definition 3.2, then G^* is said to obey* k-different match principle.

4. K-MATCH (KM) ALGORITHM

Obviously, if a released network G^* satisfies k-different match principle, given any sub-graph query Q, no adversary can identify the target with a probability higher than $\frac{1}{k}$. In this section, we propose our algorithm to find an anonymized network G^* that satisfies k-different match principle. We also prove that G^* obtained by our algorithm is a k-automorphic network and G is a sub-graph of G^*. Thus, G^* can guarantee privacy under any structural attack.

4.1 Overview

We illustrate the main ideas of KM algorithm using Figure 4. Assume that we need to guarantee that the released network G^* satisfies 2-different match (i.e. k =2). First, we partition the original network G into 2 blocks, P_{11} and P_{12}, as shown in Figure 4b. Then, we perform graph alignment on P_{11} and P_{12} to obtain two alignment blocks P'_{11} and P'_{12} (adding edge $(2,4)$), where P'_{11} is isomorphic to P'_{12}. The details of graph alignment are discussed shortly. Obviously, if a match of query Q is contained in alignment block P'_{11} (or P'_{12}), it must also be contained in the other alignment block P'_{12} (or P'_{11}). This means that, if a match of query Q can be contained in some block of the anonymized network G^*, there must exist at least 2 matches of Q in G^*.

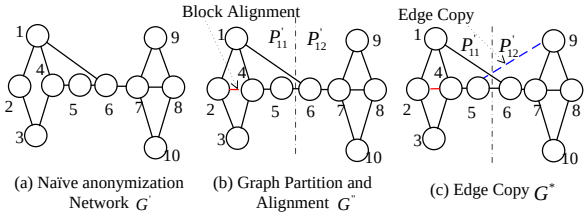


Figure 4: Graph Partition and Edge Copy

However, the anonymized network after graph partition and alignment is still insufficient to satisfy the k-different match principle. For example, given query Q_4 in Figure 2d, there is still only one match of Q_4 in the anonymized network in Figure 1b. This is because the match crosses two blocks. In order to handle the problem, we propose the "edge-copy" technique (Section 4.3).

Since block P'_{11} is isomorphic to P'_{12} under function f, for each vertex v in P'_{11}, there exists a corresponding vertex $f(v)$ in P'_{12}.

There is a crossing edge between vertex 1 (in block P'_{11}) and vertex 6 (in block P'_{12}) in Figure 4b; thus, we need to introduce another edge between vertex $f(1)$ (that is vertex 9 in block P'_{12}) and $f^{-1}(6)$ (that is vertex 5 in block P'_{11}). Figure 4c shows the anonymized network after edge-copy. Given the same query Q_4, there are two matches $m_1 = (7, 6, 9, 10, 8, 1, 5)$ and $m_2 = (4, 5, 1, 3, 2, 9, 6)$. Although these two matches have some shared vertices, they correspond to different vertices in query Q. Thus, according to Definition 3.2, these two matches are different.

The **K**-**M**atch algorithm (KM for short) in Algorithm 1 implements the approach we propose. We start by removing all identity information from the original network G resulting in the naive anonymized network G' (line 1 in Algorithm 1). We then partition G' into n blocks and cluster these blocks into m groups U_i, $(i = 1, ..., m)$, where each group U_i has at least k blocks (line 2). For each group, we perform graph alignment (we also use the term "block alignment" interchangeably) to obtain alignment blocks (lines 3-4). We replace original blocks by alignment blocks to obtain network G''(line 5). Then, we perform edge-copy to get G^*(line 6), which is reported as the anonymized network (line 7). The details of each step will be discussed shortly.

Obviously, different partitionings of the original network G and different block clusterings lead to different released networks G^* and different anonymization costs. We delay the discussion of finding a "good" partitioning and block clustering to minimize anonymization cost until Section 4.4.

According to KM Algorithm, each group U_i has k_i blocks, where $k_i \geq k$. For ease of presentation and without loss of generality, we assume in the remainder that each group has exactly k blocks.

Algorithm 1 **K**-**M**atch (KM) Algorithm

Require: **Input**: An original graph G and the parameter k.
Output: The anonymized network G^*, which is a k-auto-morphism network.
1: Generate G' by removing all identity information from G.
2: Partition G' into n blocks and cluster the blocks into m groups U_i, $(i = 1, ..., m)$, where each U_i has at least k blocks P_{ij}, $(j = 1, ..., k_i$ and $k_i \geq k)$. (see Algorithm 5 in Section 4.4)
3: **for** each group U_i **do**
4: perform graph alignment on all blocks P_{ij} in U_i to obtain alignment block P'_{ij}. (see Algorithm 2 in Section 4.2)
5: Replace each block P_{ij} by its alignment block P'_{ij} to obtain anonymized network G''.
6: For all crossing edges, perform edge-copy to obtain anonymized network G^*. (see Algorithm 4 in Section 4.3)
7: Output G^*.

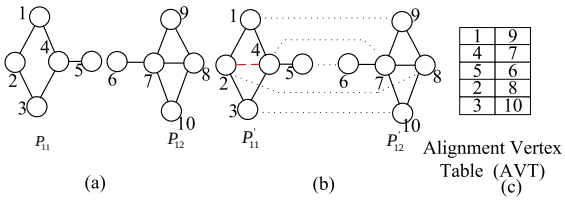
4.2 Block (Graph) Alignment

For each group U_i that is obtained following partitioning and clustering, we perform graph alignment operations on all blocks P_{ij} in U_i. Specifically, each block P_{ij} is transformed into alignment block P'_{ij}, where all P'_{ij} are isomorphic to each other. Furthermore, P_{ij} is a sub-graph of P'_{ij}.

DEFINITION 4.1. ***Alignment Vertex Instance.*** *Given a group U_i with blocks P_{ij}, $j = 1, ..., k$, assume that alignment blocks $P'_{ij} = (V'_{ij}, E'_{ij})$ are the blocks obtained after graph alignment, namely, $\forall j$ P_{ij} is a sub-graph of P'_{ij} and all P'_{ij} are isomorphic to each other.*

Due to graph isomorphism, given an alignment block P'_{ij}, for each vertex v in P'_{ij}, there must exist $k-1$ symmetric vertices in other $k-1$ blocks respectively. The set containing v and v's symmetric vertices form alignment vertex instance *I, where $|I| = k$. All alignment vertex instances are collected to form* alignment vertex table *(AVT).*

Example 2. We illustrate Definition 4.1 using Figure 5. The naive anonymized network G' in Figure 4a is partitioned into 2 blocks P_{11} and P_{12}, which are clustered into one group U_1. Figure 5 shows a graph alignment between two blocks P_{11} and P_{12} (note the addition of edge $(2, 4)$ in P'_{11} in order to get graph isomorphism between P'_{11} and P'_{12}). Since P'_{11} is isomorphic to P'_{12}, for each vertex v in P'_{11} (or P'_{12}), there exists a symmetrical vertex u in P'_{12} (or P'_{11}). Therefore, each (v, u) is an alignment vertex instance I. Note that in this example, $k = 2$; thus, each vertex v has one symmetric vertex in the other block. Figure 5 shows AVT, which contains all alignment vertex instances.



1	9
4	7
5	6
2	8
3	10

Figure 5: Block Alignment

DEFINITION 4.2. ***Alignment Cost.*** *Given a group U_i with blocks P_{ij}, $j = 1, ..., k$, assume that P'_{ij} are the blocks obtained after block alignment. The cost of block alignment in group U_i is defined as follows:*

$AlCost(U_i) = \sum_{j=1}^{k} Min(EditDist(P_{ij}, P'_{ij}))$

where $EditDist(P_{ij}, P'_{ij})$ is defined as the number of graph edit operations (insert vertex/edge, delete vertex/edge) required to transform P_{ij} into P'_{ij}.

Since P_{ij} is a sub-graph of P'_{ij}, $EditDist(P_{ij}, P'_{ij}) = |E(P'_{ij}) - E(P_{ij})|$, where $E(P_{ij})$ is a set of edges in P_{ij}. The optimal block alignment in one group U_i with k blocks means that $AlCost(U_i)$ is minimal.

THEOREM 4.1. *Finding the optimal block alignment for one group U_i with k blocks is NP-hard.*

PROOF. (Sketch) Minimal graph edit distance problem can be reduced to finding the optimal alignment. The former is a classical NP-hard problem [22]. □

Due to the hardness of block alignment, we propose an heuristic algorithm (Algorithm 2) to find a good alignment (not necessarily optimal) for k blocks in one group U_i. At the beginning of Algorithm 2, we call Algorithm 3 to build the AVT, which contains all alignment vertex instances. According to AVT, for each vertex v, we can find $k-1$ symmetric vertices in the other $k-1$ blocks. If there is an edge between v_1 and v_2 in a certain block P_{ij}, we need to introduce an edge between v_1's symmetrical vertices and v_2's symmetrical vertices in the other $k-1$ alignment blocks in the same group U_i.

Thus, the key problem is how to build AVT. Given a group U_i with blocks P_{ij}, $j = 1, ..., k$, Algorithm 3 implements our approach. Initially, we find k vertices v_{ij} from k blocks P_{ij} respectively, which have the same vertex degrees d. If there are multiple choices for d, we choose d with the largest value. If there is no vertex in one block P_{ij} with the same degree as the selected vertices in the other blocks, we choose a vertex with the largest degree in P_{ij}. The set of these k vertices form the initial alignment vertex instance. We next perform breath-first search starting from v_{ij} in each block P_{ij} in parallel. During this process, we always pair up k vertices (in k blocks P_{ij} respectively) with similar degrees. For

some vertex v_{ij} in P_{ij}, if we cannot find the corresponding vertex in the other blocks, we will introduce a dummy vertex, which has the same label as the corresponding vertex. Finally, we get an AVT containing all alignment vertex instances.

Algorithm 2 GraphAlignment: Block Alignment for k Blocks

Require: Input: A group U_i with k Blocks P_{ij}, where $j = 1, ..., k$
Output: Alignment blocks P'_{ij}, where $j = 1, ..., k$.
1: Call constructAVT algorithm (see Algorithm 3) to obtain $AVT = constructAVT(U_i)$.
2: Initialize k blocks P'_{ij}, where $j = 1, ..., k$.
3: **for** $j = 1, ..., k$ **do**
4: **for** each edge (v_1, v_2) in P_{ij} **do**
5: Insert edge (v_1, v_2) into block P'_{ij}, and edges between v_1's symmetrical vertices and v_2's symmetrical vertices in other $k-1$ alignment blocks respectively.
6: Report all alignment blocks P'_{ij}, $j = 1, ..., k$.

Algorithm 3 constructAVT(U_i), where $j = 1, ..., k$: Built AVT for a group with k blocks P_{ij}, where $j = 1, ..., k$

1: Set all vertices in each P_{ij} as "un-visited", initialize AVT
2: Find v_{ij} in each block P_{ij}, where all $degree(v_{ij}) = d$. If there are multiple choices for d, choose d with the largest value. If there are no choices for d, choose v_{ij} with the largest degree from block P_{ij} respectively
3: The set of all v_{ij} form the initial alignment vertex I instance in AVT.
4: Perform breath-first search (BFS) starting from v_{ij} in each P_{ij} in parallel.
5: During BFS, k vertices from k blocks with similar vertex degrees are collected to form an alignment vertex instance in AVT.
6: Report AVT.

The number of the iterations between lines 3-5 in Algorithm 2 depends on the total number edges in blocks P_{ij}, $j = 1, ..., k_i$. Thus, time complexity of Algorithm 2 is $O(\sum_{j=1}^{j=k_i} |E(P_{ij})|)$. Since BFS search time complexity in block P_{ij} is $O(|E(P_{ij})|+|V(P_{ij})|)$, the time complexity of Algorithm 3 is $O(Max(|E(P_{ij})|+|V(P_{ij})|))$.

After graph alignment, we replace each block P_{ij} in naive anonymized network G' (line 5 in Algorithm 1) by its alignment block P'_{ij}. In this way, we can obtain an anonymized network G''. Obviously, given a sub-graph query Q, if a match of Q is contained in a certain block P'_{ij}, Q is also contained in the other $k-1$ alignment blocks in the same group U_i. This means that Q has at least k different matches in G''. As discussed in Section 4.1, the anonymized network G'' after graph partition and block alignment still compromises some privacy. For example, if all matches of query Q crosses at least two blocks, we cannot guarantee that G'' satisfies the *k-different* match principle. Consider G'' in Figure 4b that is obtained after block alignment. Query Q_4 in Figure 2d has only one match in G''. Thus, G'' does not satisfy 2-different match principle. This can be addressed by the edge-copy technique that is described in the next section.

4.3 Edge Copy

According to line 2 in Algorithm 1, there are m groups U_i, $i = 1, ..., m$. For each group U_i, after block alignment, we have k alignment blocks P'_{ij}, ($j = 1, ..., k$). Furthermore, according to Algorithm 3, each group U_i has an AVT A_i. Union of A_i, $i = 1, ..., m$, are collected to form the overall AVT for the whole graph.

The next step is to process the crossing edges between two blocks in G''. As discussed in Section 1, the intuition behind our work is to find k-1 automorphic functions F_a. According to these functions, we can convert the original graph G into a k-automorphic graph G^*, which guarantees privacy under any structural attack.

We define function F_a ($a = 1, ..., k-1$) based on AVT. We treat alignment vertex instance I in AVT as a circularly-linked list. Specifically, we use $I.at(a)$ to denote the a-th vertex in instance I. We define $v.next$=$I.at(a+1)$, if $a \leq |I| - 1$; else $v.next = I.at(1)$, where $|I|$ is the number of vertices in I.

DEFINITION 4.3. *For each vertex v, we assume that v is in the instance I in AVT. v's automorphic function F_a ($a = 1, ..., k-1$) is defined as follows:*
1) $F_1(v)$=$v.next$;
2) $F_a(v)$=$F_{(a-1)}(v).next$, where $a > 1$.

We illustrate F_a using Figure 5. Since k=2 in Figure 5, each instance has two vertices, and we have one automorphic function. Thus, $F_1(1) = 9$ and $F_1(9) = 1$. Similarly, for vertex 4, $F_1(4) = 7$ and $F_1(7) = 4$.

In order to guarantee k-automorphism under functions F_a, we need to "copy" crossing edges which are defined as follows.

DEFINITION 4.4. ***Boundary Vertex** and **Crossing Edge.** Given a vertex v in a block P, v is a* boundary vertex *if and only if v has at least one neighbor vertex that is outside of block P. An edge $e = (v, u)$ is called a* crossing edge *if and only if v and u are boundary vertices in two different blocks.*

Assume that there is a crossing edge $e = (v_1, u_1)$. According to alignment functions, we introduce $k-1$ edges between $k-1$ pairs $(F_a(v_1), F_a(u_1))$ ($a = 1, ..., k-1$). For example, there is a crossing edge $(1, 6)$ in G' in Figure 4b. Since $k = 2$, we introduce another edge between $(F_1(1), F_1(6))$ (i.e. $(9, 5)$). This results in the network G^* (Figure 4c), which satisfies 2-different match principle. Note that, if there exists an edge between $F_a(v_1)$ and $F_a(u_1)$ in the original network G, we will not introduce an extra edge $(F_a(v_1), F_a(u_1))$.

Algorithm 4 Edge Copy Algorithm

Require: Input: The original network: G; The network after graph partition and block alignment: G''; Alignment Vertex Table: AVT
Output: The anonymized network G^*.
1: Duplicate G'' into G^* and remove all crossing edges in G^*.
2: **for** each crossing edge (v, u) in the original network G **do**
3: Add edge (v, u) and $(F_a(v), F_a(u))$ ($a = 1, ..., k-1$) into G^*.
4: Report G^* as release network.

There are $O(|E(G)|)$ edges to be copied. Thus, the number of iterations of lines 2-3 in Algorithm 4 is $O(|E(G)|)$. Therefore, the time complexity of Algorithm 4 is $O(|E(G)|)$.

4.4 Graph Partitioning

Algorithm 1 relies on the partitioning of the naive anonymized network G' into n blocks that are clustered into m groups U_i, ($i = 1, ..., m$). As noted earlier, different partitionings and block clusterings will lead to different anonymization cost $Cost(G, G^*)$(defined in Definition 2.6).

In this section, we discuss how to find a good graph partitioning and block clustering to minimize $Cost(G, G^*)$. Since G is a sub-graph of G^*, $Cost(G, G^*) = |E(G^*) - E(G)|$, namely, $Cost(G, G^*)$ indicates the number of edges introduced by the KM algorithm. According to KM, we introduce edges in both block alignment (Algorithm 2) and edge copy (Algorithm 4). The effect of graph partitioning is quite difficult to quantify. For example, if we partition G into fewer blocks, there are fewer crossing edges to be copied. On the other hand, fewer blocks imply that the size of each block is large. Thus, we need to introduce more edges to perform block alignment. The optimal graph partitioning

of G implies that KM algorithm can produce a released network G^* where $|E(G^*) - E(G)|$ is minimal. Finding the optimal graph partitioning is NP-complete, since we can reduce a NP-complete problem (graph partition with min-cut) to this problem. Thus, we instead propose a heuristic partitioning algorithm. Theorem 4.2 shows the relationship between $Cost(G, G^*)$ and graph partitioning and block clustering.

DEFINITION 4.5. *Given a group U_i with blocks P_{ij}, $j = 1, ..., k$,* anonymization cost of group U_i *is defined as follows:*

$Cost(U_i) = AlCost(U_i) + 0.5 * (k-1) * \sum_{j=1}^{k} |CrossEdge(P_{ij})|$

where $AlCost(U)$is defined in Definition 4.2 and $|CrossEdge(P_{ij})|$ is the number of crossing edges associated with block P_{ij}.

In Definition 4.5, $AlCost(U_i)$ denotes the number of edges introduced during block alignment. For each crossing edge $e = (v, u)$, we need to introduce $k - 1$ copy-edges $(F_a(v), F_a(u))$. Each crossing edge is associated with two blocks. Thus, $0.5 * (k - 1) * \sum_{j=1}^{k} |CrossEdge(P_{ij})|$ denotes the number of edges introduced into group U_i during the edge copy process.

THEOREM 4.2. *Assume that a network G is partitioned into n blocks that are clustered into m groups U_i, where each group U_i has k blocks. Let G^* be an anonymized network produced by KM algorithm. Then*

$Cost(G, G^*) = \sum_{i=1}^{m} Cost(U_i)$

where $Cost(G, G^*)$ and $Cost(U_i)$ are defined in Definitions 2.6 and 4.5, respectively.

PROOF. (Sketch) The number of edges that KM algorithm introduces equals to the sum of introduced edges in each group U_i, $i = 1, ..., m$. □

According to Theorem 4.2, we propose a greedy method to find a good graph partitioning and block clustering. In each step, we try to find the group U_i with the minimal $Cost(U_i)$. Before introducing our method, we cite the following definition.

DEFINITION 4.6. ***Frequent Sub-graph [11].*** *Given a large graph G and a minimal support min_sup, a graph g_f is called a* frequent sub-graph *of G, if and only if g_f has at least min_sup matches in G, where any two matches are edge-disjoint.*

Algorithm 5 Graph Partitioning and Block Clustering

Require: Input: The naive anonymized G' and k.
Output: a set of groups $S = \{U_i\}$, $(i = 1, ..., m)$, where each group U_i has k blocks P_{ij}, $(j = 1, ..., k)$.
1: **repeat**
2: Find frequent sub-graphs $\{g_f\}$ in G' by setting minimal support $min\_sup = k$. Find the frequent sub-graph g_f with the largest number of edges. Each match of g_f is extracted from G' as one block P_{ij}.
3: The set of all blocks P_{ij} from one group U_i'.
4: **repeat**
5: set $U_i = U_i'$.
6: **for** each block P_{ij} in U_i **do**
7: Expand block P_{ij} by one hop.
8: The set of expanded blocks form group U_i'.
9: **until** $Cost(U_i) < Cost(U_i')$
10: $G' = G' - U_i$, and insert $U_i = \{P_{ij}\}$ into answer set S.
11: **until** $|E(G)|$=0
12: Report $S = \{U_i\}, i = 1, ..., m$.

Algorithm 5 implements our graph partitioning and block clustering. First, we find all frequent sub-graphs $R = \{g_f\}$ in naive anonymized network G' by setting $min\_sup = k$. Then, we choose the frequent sub-graph g_f with the largest number of edges in R. The rationale behind the choice is that we want to partition G' into as few blocks as possible in order to minimize the number of crossing edges. All of g_f's matches are extracted from G' as the initial blocks (line 2). These blocks are clustered to one group U_i (lines 3,5). The blocks in group U_i are denoted as P_{ij}. It is straightforward to see $AlCost(U_i) = 0$, since these blocks are isomorphic to each other. If a vertex v is contained in more than one block (v is randomly arranged into one of them), we need to introduce dummy vertices for alignment, in which case $AlCost(U_i) \neq 0$. However, $AlCost(U_i)$ is still small, since these blocks have high structural similarity. According to Definition 4.5, we can get $Cost(U_i)$ at this moment. Then, we expand all P_{ij} in parallel. Let neighbor P_{ij} denote all vertices that are one hop from boundary vertices in P_{ij}. All vertices in $Neighbor(P_{ij})$ are inserted into P_{ij} (line 7). If a vertex v is contained in more than one $Neighbor(P_{ij})$, v is randomly inserted into one of them. We align the new inserted vertices from each block P_{ij} by degree similarity. We may need to introduce dummy vertices in the process. After expansion, $AlCost(U_i)$ increases, since we need to introduce edges to align the new merged vertices. If all updated blocks P_{ij} have fewer crossing edges than the original blocks, the number of introduced edges during edge copy will decrease. Block expansion continues if the overall group cost $Cost(U_i)$ decreases. Otherwise, it stops (Line 9). We remove all blocks P_{ij} in U_i from the naive anonymized network G' (line 11), i.e, $G' = G' - U_i$. Note that, we choose 'edge-based' partitioning. Specifically, all crossing edges of block P_{ij} are chosen as 'separator' to separate P_{ij} from other parts of G.

We then iterate the above process until $|E(G')| = 0$ (line 11). Note that G' is a non-labeled graph; thus, we always find frequent sub-graphs if $|E(G')| \geq k$. In the last iteration, the frequent sub-graph may be an edge or an isolated vertex.

Finally, G' is partitioned into n blocks which are clustered into m groups U_i. Each group U_i has at least k blocks.

4.5 Analysis and Discussion

Although we illustrate our methods only using sub-graph attacks, we prove that network G^* released by the KM algorithm is a k-automorphic network, which can guarantee privacy under any structural attack as defined in Definition 2.5. Lemma 4.1 is used in the main Theorem.

LEMMA 4.1. *The $k-1$ functions F_a ($a = 1, ..., k-1$) defined in Definition 4.3 are automorphic functions in the released graph G^* generated by KM algorithm. Furthermore, for each vertex v in G^*, $F_{a_1}(v) \neq F_{a_2}(v)$, where $a_1 \neq a_2$.*

PROOF. (sketch) For each vertex v and each function F_a ($a = 1, ..., k-1$), there is a mapping vertex $F_a(v)$ (see Definition 4.3). For two adjacent vertices v_1 and v_2 in G^*, there is an edge between v_1 and v_2.

If v_1 and v_2 are in the same block, according to AVT, we know that $F_a(v_1)$ and $F_a(v_2)$ are also in the same block. Due to block alignment, the two blocks are isomorphic to each other. Therefore, there is also an edge between $F_a(v_1)$ and $F_a(v_2)$.

If v_1 and v_2 are in different blocks, then edge (v_1, v_2) is a crossing edge. Due to edge-copy, there is also an edge between $F_a(v_1)$ and $F_a(v_2)$.

Therefore, for any edge (v_1, v_2), there is always an edge $F_a(v_1)$ and $F_a(v_2)$ under F_a, $a = 1, ..., k-1$. According to Definition 2.2, F_a is an automorphic function.

Furthermore, according to Definition 4.3, for each vertex v, if $a_1 \neq a_2$, $F_{a_1}(v) \neq F_{a_2}(v)$. □

THEOREM 4.3. *Given an original network G, the anonymized network G^* produced by KM algorithm is a k-automorphic network. Furthermore, G is a sub-graph G^*.*

PROOF. (sketch) According to Lemma 4.1 and Definition 3.1, we know G^* is a k-automorphic network. Furthermore, according to KM algorithm, all edges in G are preserved in G^*. Thus, G is a sub-graph of G^*. □

According to k-automorphism, we know that the released network G^* can guarantee privacy under any structural attack.

THEOREM 4.4. *For any structural attack, an adversary cannot identify the target with a probability higher than $\frac{1}{k}$ in any network G^* that is released by KM algorithm.*

PROOF. According to Lemma 4.1, we can find k automorphism in G^*. Thus, for each vertex v in G^*, there always exist k-1 symmetric vertices. We cannot distinguish v from its other k-1 symmetric vertices based on any structural information. Therefore, for any structural query, the released network G^* produced by KM algorithm always contains at least k symmetric vertices, or contains null. Thus, an adversary cannot identify the target with a probability higher than $\frac{1}{k}$. □

There is a well-known trade-off between "privacy" and "accuracy" for all anonymization techniques. As the network is modified to achieve anonymization, it diverges from its original form due to the addition of vertices and edges. Therefore, there is a divergence of the results of an analysis on G^* from that on G. This inaccuracy occurs at varying degrees. In this work, we want to preserve all edges in the original network and guarantee privacy under any structural attack. To achieve this, KM algorithm introduces new edges. Actually, according to KM algorithm, the number of edges that we introduce to achieve k-automorphism is bounded to $(k-1) * |V(E)|$, where $|V(E)|$ is the number of edges in the original graph G.

There exist many non-trivial automorphism partitions of vertices in real networks [20]. "Automorphism partition" is similar to the concept of "AVI" (Definition 4.1) in our paper. An AVI is an "automorphism partition" with no less than k vertices. Essentially, it is only necessary to insert noisy edges to handle "automorphism partition" with fewer than k vertices. The high symmetry property [20, 12, 17] of real networks guarantees that KM algorithm introduces fewer noisy edges to satisfy k-automorphism model and the worst case bound is unlikely to be reached.

In theory aspect, the utility of KM algorithm depends on how symmetrical the original graphs are. If there are many automorphism partitions [20] with no fewer than k vertices in original graph, we will introduce few noisy edges. The utility of KM will be good in this case; otherwise, the utility will degrade. Many real networks are known to have high symmetry property [20, 12, 17].

5. DYNAMIC RELEASES

So far, we only consider "one-time" release of network data. However, this is not satisfactory for evolutionary networks and dynamic social network analysis [2, 10]. In these cases, it is necessary to re-publish network data periodically. This poses additional challenges. Specifically, even though each publication may satisfy k-automorphism, an adversary can still identify the target with a high probability when a sequence of releases of the network are analyzed together. Therefore, to support dynamic network analysis, it is necessary to be able to find connection between multiple releases of the network while ensuring privacy (specifically, the enforcement of k-automorphism) in these releases. These goals are usually conflicting and introduce complications.

Example 3 As shown in Figure 6, given a social network G, we need to publish G at times T_1 and T_2. According to KM algorithm, we get two released networks at times T_1 and T_2, denoted as G_1^* and G_2^*, respectively. Although both G_1^* and G_2^* individually satisfy 2-automorphism, an adversary can still identify target Bob uniquely. Assume that an adversary knows that sub-graph Q_4

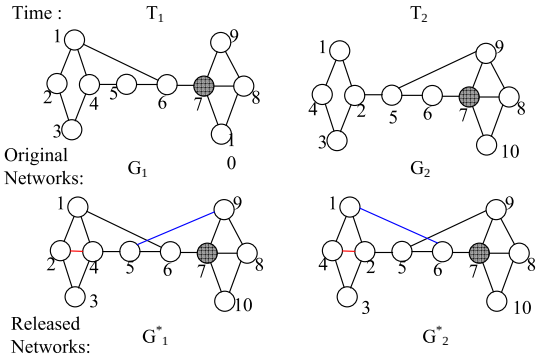


Figure 6: Dynamic Release

(see Figure 2d) exists around target Bob at both time T_1 and T_2. According to sub-graph attack Q_4 at time T_1, an adversary knows that there are two candidates vertices $\{4, 7\}$ for Bob in G_1^*. Similarly, at time T_2, there are still two candidates $\{2, 7\}$ for Bob. Since Bob exists at both T_1 and T_2, it is straightforward to determine that vertex 7 corresponds to Bob.

To remedy the above problem, a naive method is to remove all vertex IDs, or permute vertex IDs randomly (namely, a given vertex ID does not correspond to the same entity in different publications). However, if all vertex IDs are removed or vertex IDs are randomly permuted, we cannot locate the same individuals in different publications of this network. It becomes impossible to conduct proper data analysis. In this section, we propose *vertex ID generalization* technique to reduce the risk of identifying the target in dynamic releases, while preserving vertex IDs as much as possible.

5.1 Vertex ID Generalization

For ease of presentation, in this subsection, we assume that there are no vertex insertions or deletions in different releases of the network. This means that the set of all vertex IDs remains unchanged in different publications of the same network.

Recall that we define $k-1$ functions F_a $(a = 1, ..., k-1)$ in Definition 4.3 based on AVT, which are $k-1$ automorphic functions in G^* (see Lemma 4.1). Given a structural query Q, if a vertex v satisfies query Q, vertices $F_a(v)$ (defined in Definition 4.3), $a = 1, ..., k-1$, also satisfy Q. Thus, we can get at least k results for any structural query Q.

DEFINITION 5.1. *For a vertex v in a released network G_t^* at time T_t, the set of v's corresponding vertices $F_a(v)$, $a = 1, ..., k-1$, and v are called v's* associated query result set *at T_t, denoted as $Res(v, G_t^*)$.*

It is straightforward to show $|Res(v, G_t^*)| = k$, where G_t^* is generated by the KM algorithm at time T_t.

LEMMA 5.1. *Given a series of publications $\Omega = \{G_t^*\}$ of a network G, $(t = 1, ..., s)$, a vertex v cannot be identified with a probability higher than $\frac{1}{k}$, if the following holds:*

$$|Res(v, G_1^*) \cap Res(v, G_2^*) \cap ... \cap Res(v, G_s^*)| = k$$

We illustrate Lemma 5.1 by Example 3 in Figure 7, where $k = 2$. For vertex 1, according to AVT A_1 and A_2, $F_1(1) = \{9\}$ in both G_1^* and G_2^*. Thus, $Res(1, G_1^*)$ = $Res(1, G_2^*)$= $\{1, 9\}$. An adversary cannot distinguish these two vertices 1 and 9. However, for vertex 7 in the network, $F_1(7) = \{4\}$ in G_1^* and $F_1(7) = \{2\}$

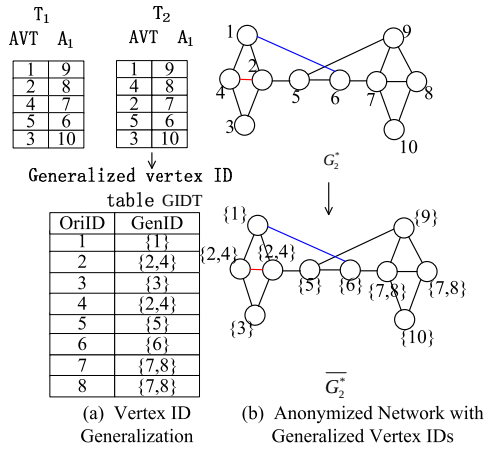


Figure 7: Vertex ID Generalization

in G_2^*. Therefore, $Res(7, G_1^*)$=$\{4, 7\}$ and $Res(7, G_2^*)$ =$\{2, 7\}$. If an adversary knows that the target exists at both times T_1 and T_2, and launches sub-graph attack Q_4 (in Figure 2d) to G_1^* and G_2^*, it is easy to locate the target as vertex 7. The following lemma gives the sufficient condition for a vertex v to be unidentifiable from a series of publications of the same network.

LEMMA 5.2. *Given a series of publications $\Omega = \{G_t^*\}$ of a network G, $(t = 1, ..., s)$, a vertex v cannot be identified with a probability higher than $\frac{1}{k}$, if the following holds:*

$Res(v, G_1^*) \cap Res(v, G_2^*) \cap ... \cap Res(v, G_s^*) = Res(v, G_1^*)$

where $|Res(v, G_1^*)| = k$.

Lemma 5.2 shows the intuition of the proposed "vertex ID generalization" method in dynamic releases. First, we illustrate our method through Example 3. Assume that we have obtained 2-automorphic networks G_1^* and G_2^* produced by KM algorithm (Figure 6) along with AVT A_1 and A_2 (Figure 7). Based on A_1, we can define $k-1$ automorphic functions in G_1^*. Similarly, we can define other $k-1$ automorphic functions in G_2^* based on A_2. Then, we generate an anonymized network $\overline{G_2^*}$, where each vertex in $\overline{G_2^*}$ is assigned a *generalized vertex ID* (see generalized vertex ID table GIDT in Figure 7). For each record r in GIDT, $r.OriID$ is its original vertex ID, and $r.GenID$ is its generalized vertex ID. Initially, each generalized vertex ID has only one element, that is $r.GenID = \{r.OriID\}$.

In order to protect released networks in dynamic releases, we perform the following vertex ID generalization process. We know $Res(7, G_1^*) = \{4, 7\}$ and $Res(7, G_2^*) = \{2, 7\}$. In order to avoid the disclosure of vertex 7, according to Lemma 5.2, we need to guarantee that $Res(7, G_1^*) \cap Res(7, G_2^*) = Res(7, G_1^*) = \{4, 7\}$. Thus, in the released network $\overline{G_2^*}$, we need to insert 4 into 2's generalized vertex ID, that is $2.GenID = \{2, 4\}$. Figure 7 shows all generalized vertex IDs at time T_2. To release anonymized network at time T_2, for each vertex v, we attach $v.GenID$ as its vertex ID.

The GenID algorithm in Algorithm 6 implements our approach, which only depends on the two AVTs A_1 and A_t, and the anonymized networks G_1^* and G_t^* (obtained by KM algorithm at times T_1 and T_t). First, we initialize the generalized vertex ID table GIDT (line 1 in in Algorithm 6). We define $k-1$ automorphic functions F_a^1 in G_1^*, $a = 1, ..., k-1$, based on A_1 (line 2). Similarly, we define $k-1$ automorphic functions F_a^t in G_t^*, $a = 1, ..., k-1$, based on A_t (line 3). Then, for each vertex v in G_t^*, if $F_a^1(v) \neq F_a^t(v)$, $a = 1, ..., k-1$, we insert vertex ID $F_a^1(v)$ into $F_a^t(v)$'s generalized vertex ID, that is $F_a^t(v).GenID$ (lines 4-7). Next, for each vertex v in G_t^*, we replace the vertex ID $v.OriID$ by its generalized ID $v.GenID$ to get network $\overline{G_t^*}$ (lines 8-9). We publish $\overline{G_t^*}$ as a released network at time T_t (line 10).

After GenID algorithm, each network G_t^* is transformed into a k-automorphic network $\overline{G_t^*}$ with generalized vertex IDs. The following theorem proves that $\overline{G_t^*}$ will not leak any privacy under dynamic releases. Notice that, at time T_1, $\overline{G_1^*}$ is the same as G_1^*. We do not distinguish $\overline{G_1^*}$ and G_1^* in the following discussion.

Algorithm 6 Generalize Vertex ID For Released Network G_t^*

Require: Input: AVT A_1 for the network G_1^*, and AVT A_t for the network G_t^*.
Output: The anonymized network after vertex ID generalization: $\overline{G_t^*}$.
1: Initialize table GIDT.
2: Based on A_1, define $k-1$ automorphic functions F_a^1 in G_1^*, $a = 1, ..., k-1$.
3: Based on A_i, define $k-1$ automorphic functions F_a^t in G_2^*, $t = 1, ..., k-1$.
4: **for** each vertex v in G_t^* **do**
5: **for** $a = 1, ..., k-1$ **do**
6: **if** $F_a^1(v) \neq F_a^t(v)$ **then**
7: Insert $F_a^1(v)$ into $F_a^t(v).GenID$.
8: **for** each vertex v in G_t^* **do**
9: Replace $v.OriID$ by its generalized vertex ID $v.GenID$.
10: Report $\overline{G_t^*}$.

THEOREM 5.1. *Given a series of publications $\Omega = \{\overline{G_t^*}\}$ $(t = 1, ..., s)$, where $\overline{G_t^*}$ is generated by GenID algorithm at time T_t, it is not possible to identify v in Ω with a probability higher than $\frac{1}{k}$.*

PROOF. (sketch) Consider any publication $\overline{G_t^*}$ at time T_t, $t \neq 1$. According to GenID algorithm, for each v in $\overline{G_t^*}$, we know that $Res(v, \overline{G_t^*}) \supseteq Res(v, \overline{G_1^*})$. Thus, $Res(v, \overline{G_t^*}) \cap Res(v, \overline{G_1^*}) = Res(v, \overline{G_1^*})$. Therefore, $Res(v, G_1^*) \cap Res(v, G_2^*) \cap ... \cap Res(v, G_s^*) = Res(v, G_1^*)$. According to Lemma 5.2, Theorem 5.1 holds. □

In Section 2.2, we stated that we use anonymization cost to measure the information loss. In fact, the anonymization cost can nicely estimate the structural information loss since it counts the number of edges that we change before and after anonymization. However, the anonymization cost is not suitable for measuring the information loss of vertex ID generation, since this generalization relates to the content of a vertex ID. Therefore, we propose a new measure, called *average generalized vertex ID size*, to quantify the information loss caused by vertex ID generalization.

DEFINITION 5.2. *Given a released network $\overline{G_t^*}$ produced by* GenID algorithm*, average generalized vertex ID size, denoted by $AvgIDSize(\overline{G_t^*})$, is defined as follows:*

$$AvgIDSize(\overline{G_t^*}) = \frac{\sum_{v \in V(\overline{G_t^*})} |v.GenID|}{|V(\overline{G_t^*})|}$$

where $V(\overline{G_t^*})$ is the set of vertices in $\overline{G_t^*}$.

The above definition indicates that the larger the $AvgIDSize(\overline{G_t^*})$, the higher is the information loss. We prove that the information loss caused by *GenID algorithm* is bounded.

THEOREM 5.2. *For any publication $\overline{G_t^*}$ at T_t, where $\overline{G_t^*}$ is produced by GenID algorithm, the following holds*

$$1 \leq AvgIDSize(\overline{G_t^*}) \leq k$$

PROOF. Proof is based on Algorithm 6. Due to space limitation, we omit the details. □

The above theorem implies that the information loss in dynamic releases can be controlled by k.

5.2 Vertex Insertion and Deletion

In practice, some vertex insertions and deletions will occur in different publications G_t^*. If an adversary knows that the target exists in G_1^* but not in G_2^*, it is easy to identify the target. We propose the following techniques to handle this problem. There are two cases:

1) (Deletion) There is a vertex ID v that exists in G_1^*, but not in $\overline{G_t^*}$, where $t \neq 1$. In this case, we find an arbitrary vertex ID u that exists in both G_1^* and $\overline{G_t^*}$. We insert v into u's generalized vertex ID (i.e. $u.GenID$) in $\overline{G_t^*}$.

2) (Insertion) There is a vertex ID v that exists in G_t^* but not in G_1^*. Assume that instance I contains v in AVT A_t. For each vertex u in I, we insert v into $u.GenID$ in $\overline{G_t^*}$. This means that the new vertex ID exists in at least k vertices in $\overline{G_t^*}$.

Although the above method only considers the released network G_1^* at time T_1 and $\overline{G_t^*}$ at time T_t, an adversary still cannot identify the target based on $\overline{G_{t_1}^*}$ and $\overline{G_{t_2}^*}$, where $1 < t_1 < t_2$.

THEOREM 5.3. *Based on any two historical publications $\overline{G_{t_1}^*}$ and $\overline{G_{t_2}^*}$ ($1 < t_1 < t_2$), an adversary cannot identify the target with probability higher than $\frac{1}{k}$ even if the adversary knows the information about insertion or deletion of a vertex.*

PROOF. Proof is based on our insertion and deletion method. Due to space limitation, we omit the details. □

THEOREM 5.4. *Consider the publication $\overline{G_t^*}$ at T_t. The number of vertices that exist in T_t but not in T_1 is denoted by $|Ins|$; the number of vertices that exist in T_1 but not in T_t is denoted by $|Del|$. The average generalized vertex ID size in $\overline{G_t^*}$ has the following bounds.*

$$1 \leq AvgIDSize(\overline{G_t^*}) \leq k + \frac{(k-1)\times|Ins|}{|V(G_t^*)|} + \frac{|Del|}{|V(G_t^*)|}$$
$$\leq 2k - 1 + \delta$$

where $\delta = \frac{|V(G_1^*)|}{|V(G_t^*)|}$.

PROOF. Proof is based on our insertion and deletion method. Due to space limitation, we omit the details. □

Usually, in evolutionary networks and dynamic social network analysis, the number of inserted and deleted vertices (i.e. $|Ins|$ and $|Del|$) is very small, compared with the number of vertices in the original network (i.e. $|V(G_1^*)|$).

6. EXPERIMENTS

In this section, we evaluate our methods using both synthetic and real data sets, and compare them with the existing state-of-the-art techniques, such as k-degree anonymity [13], 1-neighborhood-diversity [23] and graph generalization method [7]. These methods are denoted as "Against-Degree" [23], "Against-1Neighbor" [13] and "Generalization" [7] respectively. All of the methods have been implemented using standard C++. The experiments are conducted on a P4 3.0GHz machine of 1G RAM running Windows XP.

6.1 Datasets

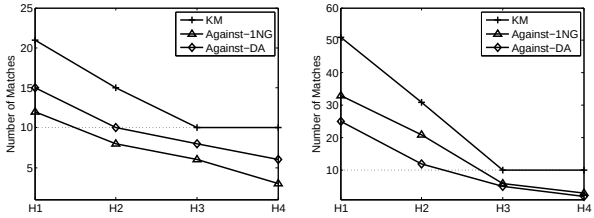
1) **Real Datasets**: *Prefuse graph:* (*http://prefuse.org/*). This dataset has 129 nodes and 161 edges.

Co-author graph: (*http://liinwww.ira.uka.de /bibliography*). The dataset consists of 7955 authors of papers in database and theory conferences. It has 10055 edges.

2) **Synthetic Datasets**: We use software Pajek (*http:// vlado.fmf.uni-lj.si/pub/networks/pajek/*) to generate two kinds of random graphs. The default number of vertices in synthetic dataset is 1000.

Erdos Renyi Model: This is a classical random graph model. It defines a random graph as N vertices connected by M edges that are chosen randomly from the $N(N-1)/2$ possible edges. In our experiments, we set N=1000 and M=5000. This dataset is denoted as "ER".

Scale-Free Model: A scale-free network is a network whose degree distribution follows a power law, at least asymptotically, i.e. for degree d, its probability density function is $P(k) = d^{-\gamma}$. Usually, 2.0< γ <3.0. In our experiments, we set the number of vertices to be 1000 and γ=2.5. This dataset is denoted as "SF".



(a) in Co-author network (b) in Prefuse network

Figure 10: Number of Matches in Vertex Refinement Attack

6.2 Power of Privacy Protection

We evaluate released networks G^* produced by the KM algorithm under different attacks. The default value of k is 10. First, for any degree d in released network G^*, we report the frequency of vertices with degree d in G^*. As depicted in Figure 8, the minimal frequency is 10. This means that G^* can guarantee privacy under degree attack.

We test our method in sub-graph attacks. We assume that an adversary knows some local structure around a target. We randomly extract some sub-graphs from original network G as query graphs. Figure 9 shows the number of matches of Q in released networks G^*. With increasing $|E(Q)|$ (i.e. the number of edges in Q), the number of matches decreases. When $|E(Q)| > 12$ in Figure 9a in Prefuse dataset, there are less than 10 matches in released networks produced by "Against-DA" and "Against-1NG". This means that there are privacy leaks in these released networks. The reason is that these two algorithms only consider a single type of attack, i.e. degree-attack or 1-neighbor-graph attack, respectively. Due to k(=10)-automorphism in our released network, given any sub-graph query, the number of matches is never less than 10. We obtain similar results in co-author dataset and synthetic datasets, as shown in Figure 9.

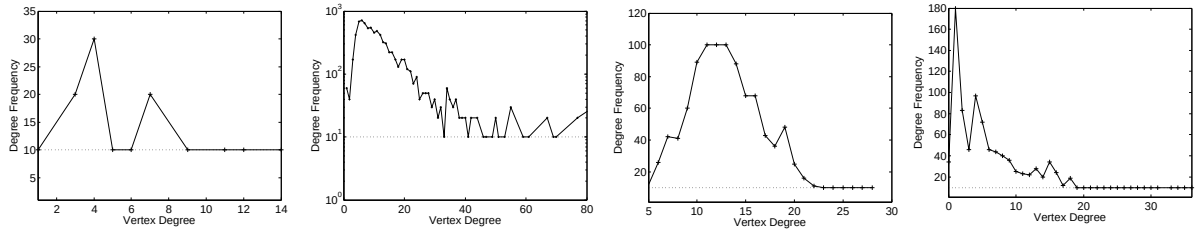
These results demonstrate that existing methods cannot be resilient to multiple attacks. Due to k-automorphism in our released network, an adversary cannot distinguish a vertex from the other $k-1$ symmetric vertices. Thus, our method can guarantee privacy under any structural attack.

Similarly, under vertex-refinement attack in Figure 10, our method can guarantee privacy; however, methods in [23] and [13] cannot resist vertex-refinement attack. Due to space limitation, we omit some of the results, such as the results on other datasets and the results under hub-fingerprint attacks. However, all results confirm the strong power of our privacy protection method.

We do not compare our method with graph generalization method [7]. For this method, we need to generate different sample instances from the generalized network. Individual sample instances cannot resist structural attacks. Generalization method protects the private information by introducing "uncertainty". Specifically, users cannot know which sample instance corresponds to the original network. As discussed earlier, "uncertainty" also limits its use in practice. For example, it cannot answer queries that find the shortest-path distance between two given vertices.

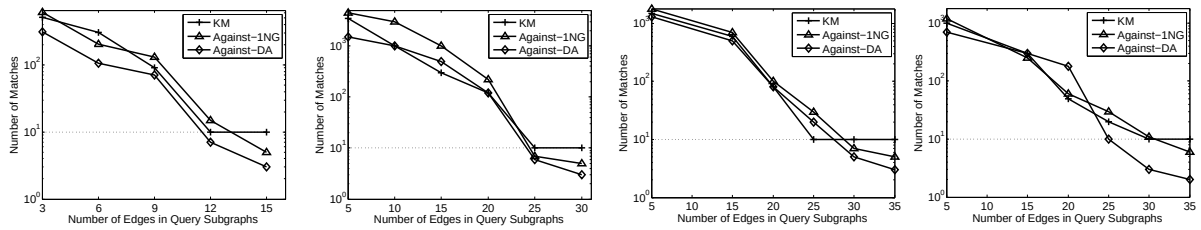
6.3 Utility Evaluation

We test G^* produced by the KM algorithm using three statistical



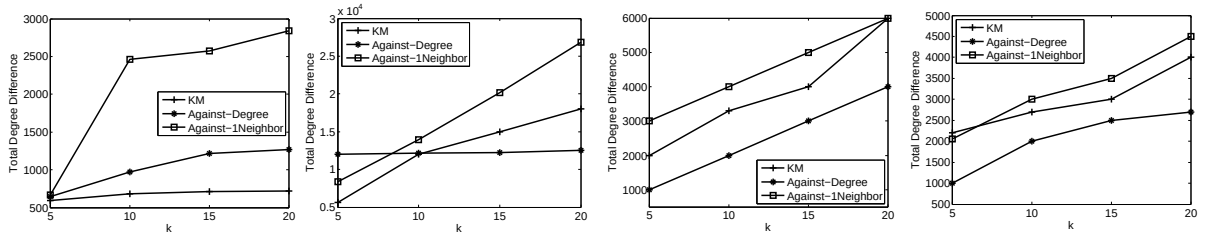
(a) in Prefuse network (b) in Co-author network (c) in ER dataset (d) in SF dataset

Figure 8: Degree Frequencies in Released Networks



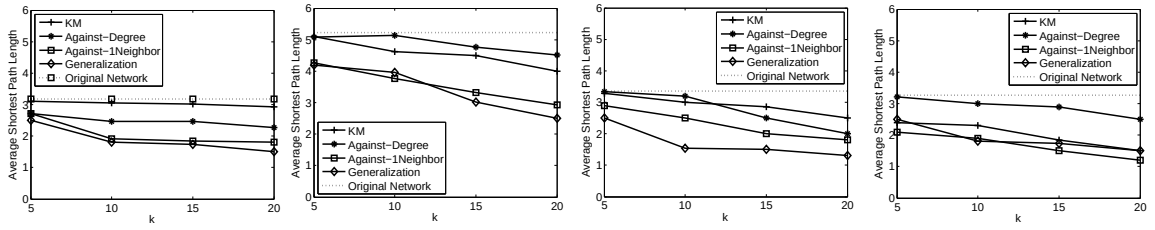
(a) in Prefuse network (b) in Co-author network (c) in ER network (d) in SF network

Figure 9: Number of Matches in Sub-graph Attack



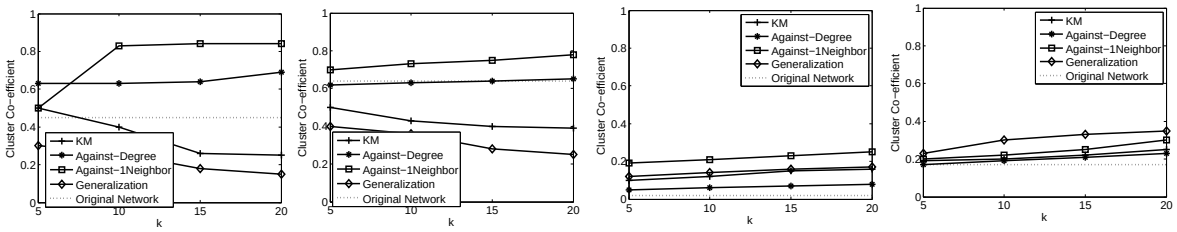
(a) in Prefuse network (b) in Co-author network (c) in ER network (d) in SF network

Figure 11: Total Degree Differences



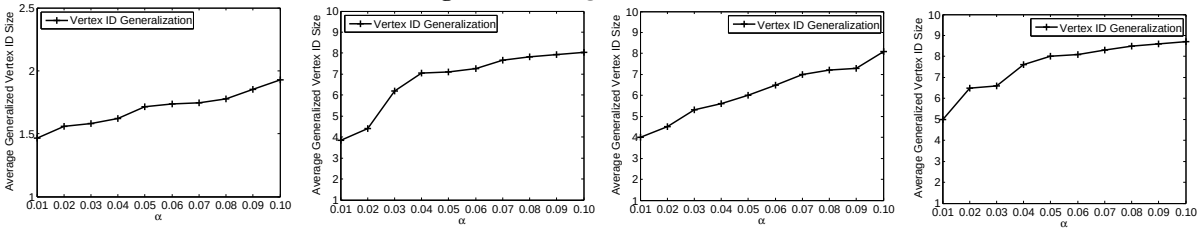
(a) in Prefuse network (b) in Co-author network (c) in ER network (d) in SF network

Figure 12: Average Shortest Path Length



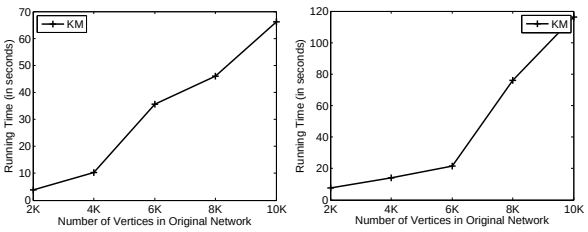
(a) in Prefuse network (b) in Co-author network (c) in ER network (d) in SF network

Figure 13: Average Cluster Co-efficient



(a) in Prefuse Dataset (b) in Co-author Dataset (c) in ER Dataset (d) in SF Dataset

Figure 15: Average Generalized Vertex ID Size



(a) Erdos Renyi Model (b) Scale Free Model

Figure 14: Running Time vs. $|V(G)|$

measures: (TDD) in Figure 11, average shortest-path length (ASP) in Figure 12, and average cluster co-efficient (ACC) in Figure 13. Note that, the dashed lines in Figures 12 and 13 denote the values of ASP and ACC in original networks, which do not changes with increasing of k.

We also compare the KM algorithm with the "Against-Degree" [23], "Against-1Neighbor" [13] and "Generalization" [7] methods. We vary k from 5 to 20. In "Against-Degree", there are a number different ways to generate k-degree anonymous network. We use DP+Supergraph method, which is reported as optimal [23]. In "Generalization", we randomly generate 200 samples for utility test, which is the same as the methods in [7].

We do not compare our method with graph generalization [7] in TDD. In graph generalization method, samples are generated based on the degree information in published generalized network. Thus, TDD in graph generalization method [7] is 0. Our method outperforms graph generalization method in the two other measures, ASP and ACC. Furthermore, the utility of the generalized network depends on the generated samples. If the samples are similar to the original network, the utility will be good. However, samples are randomly generated from a large number of possible worlds. Thus, it is difficult to generate similar samples with the original network.

As opposed to the "Against-Degree" and "Against-1Neighbor" algorithms, KM algorithm does not guarantee the best utility of released networks. For example, although TDD of KM is the least in Prefuse dataset, this not the case in co-author dataset. However, both "Against-Degree" and "Against-1Neighbor" can only resist a single type of attack. The released networks produced by "Against-Degree" and "Against-1Neighbor" may leak private information under some other structural attacks, for example, sub-graph attack in Figure 9.

Generally speaking, the utility value of our method is better than "Against-1Neighbor", and not as good as "Against-Degree". However, our method does not assume a single type of attack, and protects released network from multiple structural attacks. Therefore, it provides much stronger privacy protection than the others. Strong privacy protection and good utility are always conflicting goals.

6.4 Scalability Test

We test the scalability of KM algorithm in synthetic datasets. We vary $|V|$ (the number of vertices) from 2000 to 10,000. The default value of $k = 10$. Figure 14 shows that our algorithm has good scalability.

6.5 Performance Under Dynamic Releases

We also tested our vertex ID generalization with default $k = 10$. Given two original networks G_1 and G_t at times T_1 and T_t, the change ratio α is defined as $\alpha = \frac{|E(G(i))-E(G_1)|}{|E(G_1)|}$. In order to generate G_t, we randomly delete $0.5 \times \alpha \times |E(G_1)|$ edges from G_1, and then sequentially add $0.5 \times \alpha \times |E(G_1)|$ random edges. We vary α from 0.01 to 0.1. We define average generalized vertex ID size in Definition 5.2 to evaluate information loss in dynamic releases. The average generalized vertex ID size increases with α(Figure 15). We change 10% of the edges in G_1, the average generalized vertex ID size is 1.93 in Prefuse data and 8.03 in co-author data. However, the increase of average generalized vertex ID size is bounded (Theorems 5.2 and 5.4).

7. CONCLUSIONS

Due to increasing social network applications, privacy concerns in social networks have become increasing important. In this paper, we propose a systematic model, called k-automorphic network, to avoid identity disclosure in released networks. Given an original network G, our algorithm converts G into a k-automorphic network G^*, which is then published. Our method can guarantee privacy under any structural attack. In order to support dynamic social network analysis, we also address dynamic releases of the network. It is straightforward to push attribute generalization method [16] into KM algorithm to handle networks with vertex attributes (i.e. labeled graphs).

8. REFERENCES

[1] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, 2007.

[2] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, 2006.

[3] Y. Bu, A. W.-C. Fu, R. C.-W. Wong, L. C. 0002, and J. Li. Privacy preserving serial data publishing by role composition. *PVLDB*, 1(1).

[4] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *PVLDB*, 1(1).

[5] K. B. Frikken and P. Golle. Private social network analysis: how to assemble pieces of a graph privately. In *WPES*, 2006.

[6] G. Ghinita, Y. Tao, and P. Kalnis. On the anonymization of sparse high-dimensional data. In *ICDE*, 2008.

[7] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural reidentification in anonymized social networks. In *VLDB*, 2008.

[8] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava. Anonymizing social networks. *Technical report, University of Massachusetts Amherst*, 2007.

[9] R. J. B. Jr. and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, 2005.

[10] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *KDD*, 2006.

[11] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11(3), 2005.

[12] J. Lauri and R. Scapellato. *Topics in Graph Automorphisms and Reconstruction*. Cambirdge University Press.

[13] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.

[14] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[15] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *PODS*, 1998.

[16] L. Sweene. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), 2002.

[17] H. Wang, G. Yan, and Y. Xiao. Symmetry in world trade networks. *Journal of Systems Science and Complexity*, 2008.

[18] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, 2006.

[19] X. Xiao and Y. Tao. M-invariance: towards privacy preserving re-publication of dynamic datasets. In *SIGMOD Conference*, 2007.

[20] Y. Xiao, M. Xiong, W. Wang, and H. Wang. Emergence of symmetry in complex networks. *Physical Review E*, 77(6), 2008.

[21] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, 2008.

[22] K. Zhang, J. T.-L. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs and related problems. In *CPM*,

1995.

[23] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, 2008.